User Guide

OpenAKC

Version 1.0 ◆ 30 August 2020

Table of Contents

WARNING: This is a STUB/Incomplete Document… it contains notes, and examples which will eventually make up the OpenAKC Administrator Guide.

1   Introduction

1.1 Purpose

OpenAKC is in essence an SSH public key manager for Linux, implemented via a Client/Server API so that all public keys are held centrally. Additionally it adds advanced features such as session recording and powerful controls over how/when the keys are used, including the ability to restrict the root user after login.

Functionality including, but not limited to:-

*   Centrally manage "static trust", entirely replacing (and extending) the "authorized_keys" functionality.

*   Allow users to register personal ssh key pairs, and then be granted access based on directory information such as AD, LDAP or even local Unix accounts, without requiring directory access on each host in the environment (Eg. using bastion/jump hosts).

*   Do session recording without requiring a "proxy" or "man-in-the-middle" to access the session data.

*   Extend the traditional restrictions which are applied to keys, permitting time/day/date based restrictions etc.

*   Manipulate Linux capabilities to restrict functions available to the "root" even after they log in, or to restrict functions gained by using "sudo".

*   All security configuration performed in a central security server (or cluster of servers) via an API, no access to the managed hosts is required by OpenAKC admins after the client is installed.

1.2 Scope

This document describes the operation of OpenAKC including installation, configuration, operation and troubleshooting. It also touches on some details of the low level operation of the system, as well as recommended OpenSSH configuration to make the most of OpenAKC features.

1.3 Versioning

The versioning of OpenAKC follows a number scheme, of *major* dot *minor* dot *patchlevel*, where odd numbered *minor* versions are development releases, and the *major* version is updated whenever a change is made which may result in existing configuration being broken or needing to be updated.

Eg, 1.0.15 is a stable release, while 1.1.15 is a development release.

1.4 System Components

The system consists of 3 packages, containing the client, server and API tools components.

"openakc-server" – this is the backend security server which implements the API

"openakc-tools" – this is the API tool, used by everyone to interface with the system.

"openakc" – this is the client package installed on the systems to be managed by OpenAKC.

2   Software Installation

Depending on your environment, you may wish to install/update directly from the main repo, or download the packages for use either in a standalone environment, or in a private or corporate repository so that new versions can be tested.

2.1 Installing via OS repo.

The use of "sudo", or the root account is assumed where necessary.

2.1.1 Ubuntu/Debian

Download and add the repository key:

```
wget -O - https://raw.githubusercontent.com/netlore/OpenAKC/master/resources/openakc.key | apt-key add -
```

Add the repository:

```
echo "deb https://netlore.github.io/OpenAKC/repos/ubuntu/20.04 ./" > /etc/apt/sources.list.d/openakc.list
```

Update packages:

```
apt update
```

Install packages, depending on the requirements, using "apt install <packagename>"

2.1.2 Redhat/Centos/Fedora

2.1.2.1 Older systems using YUM

Add the repository:-

```
wget -O - https://netlore.github.io/OpenAKC/repos/openakc-el7.repo > /etc/yum.repos.d/openakc.repo
```

Install packages, depending on the requirements, using "`yum install <packagename>`"

2.1.2.2 Newer systems using DNF

Add the repository (depending on OS, you may first have to install the config-manager):-

```
dnf install 'dnf-command(config-manager)'
dnf config-manager --add-repo https://netlore.github.io/OpenAKC/repos/openakc-el8.repo
```

Install packages, depending on the requirements, using "`dnf install <packagename>`"

2.1.3 OpenSuSE/SuSE Enterprise

A SuSE is currently planned, but not yet available, you can use the existing packages, but you must ensure that the package "libcap-progs" is installed, and you may have to ignore a dependency on "openssh-clients".

Packages tested with OpenSuSE 15+

```
wget -O - https://netlore.github.io/OpenAKC/repos/openakc-opensuse15.repo > /etc/zypp/repos.d/openakc.repo
```

2.2   Initial Setup

2.2.1 Server Installation

2.2.1.1 Combined jump host/security server

The package "openakc-server" provides the server itself, which handles authentication requests, registration requests from the client package, and permissions updates etc.

The package "openakc-tools" will also be installed, and contains the interface tool, which allows users to register their personal keys, and also allows administrators to perform administration tasks such upload, replace, or delete static keys and configure rules etc. The "openakc-tools" package can also be installed stand-alone on a separate system either as a tool for administration, or to facilitate a configuration where users register keys from different hosts where "openakc-server" is not installed.

Running on the same host as the "openakc-server" package, to verify that the server is working after initial installation, it is possible to type "openakc ping" after installing the server packages. This should force the server to write out a default configuration and reply with "OK: Pong!".

2.2.1.2 Separated jump host/security server

In the case where you wish to have a separate security server, then the "openakc-tools" package can be installed on a 3rd system, if normal users need to register personal keys from that 3rd system, then the key generated in "/var/lib/openakc/keys" by the "openakc-tools" package, can be copied to the same folder on the security server. The only restriction is that the two systems must use the same directory, such that users, group memberships etc. are identical on both.

If you only wish to manage trust relationships, not dynamic users then there is no need for any system which users can connect to, administration can be performed locally or from another machine with the "openakc-tools".

2.2.2 Configuring an OpenAKC administrator

After running the "openakc" tool, for the first time, a folder will be created in the users home folder, called "~/.openakc", and an openssl key pair will be created. If the public key from this folder is copied to the "/var/lib/openakc/keys" folder on the server, then this will permit the user to perform security administration tasks.

If the "openakc-tools" package is installed on a machine other than the server, then a configuration file will be required to define the APIS variable, and this can be located centrally in "/etc/openakc/openakc.conf", or in the users home folder, eg "~/.openakc/config"

2.2.3 Clustering the security server

Clustering multiple security servers can be achieved using some form of shared or replicated storage, perhaps a NAS solution using NFS, file replication (such as GlusterFS) or some form of clustered storage with a shared backend such as a SAN. Simply replicating the "/var/lib/openakc" folder across all the nodes in the cluster will allow any number of nodes to work together.

2.2.4 Client Installation

To set up your clients, install the package "openakc" on each host, and ensure that the client config (/etc/openakc/openakc.conf) lists the appropriate OpenAKC API server(s) in the APIS field (this can be either name or IP). The configuration is pre-populated with "openakc01 & openakc02", so you could simply ensure that these values are configured in DNS, (or in the /etc/hosts on the client for testing), or alter them as you prefer.

Once the client is installed, and configured, it should be possible to run "openakc-plugin ping", to get an "OK: Pong!" response from the server, confirming successful communication. The plugin has only very minimal command line functions, but if for any reason you rebuild the server and loose it's openssl identity or a server which clients have already contacted has it's identity changed (perhaps during setting up of clustering), you can remove the server identity from a client using "openakc-plugin resetkeys". The client will then re-establish communication with the server.

3 Configuring OpenAKC

Functions

Keystroke logging.

Keystroke logging can be a powerful tool for security, and for understanding what went wrong when an issue occurs.

Each session opened via OpenAKC will use the same API it used to authenticate via the security server, to log some details of the connection, and assuming session recording is not disabled, it will send a record of what was displayed on the terminal, including the users actions.

This session recording is very granular (you can break down recordings by host, or by user… even tie a recording to a specific session, or PTY), and can record sessions that do not come via a specific bastion/jump host, as they are recorded at the destination and streamed back to the OpenAKC server. However, be aware however that the recordings are not 100% real time, as there is some local buffering for encryption and network reasons, otherwise the users session performance would be impacted. We therefore recommend that in high security environments, or where there may be a legal requirement to retain session recordings ,that you always use "defense in depth", consider channelling all ssh connections via a bastion/jump host, or other choke point where session recording can also take place. The tools will then complement each other.

If you are using RedHat, for example, you could use SSSD to provide access to AD/LDAP on the bastion/jump host and security server (which OpenAKC will need), and then use tlog to record users sessions on the baston/jump host [1]. These will be much less granular, ultimately more difficult to use, since a single session on the jump host may encompass many connections to different machines and indeed may not always record what you need if there is trust between hosts inside the environment, but it is possible you may catch something which OpenAKC does not. For example, if a user does something which causes an immediate kernel panic, then some part of the session log may never be flushed from the buffers. Equally, some failure on the bastion or jump host (or a user who is able to access without going via the choke point) might cause it to miss events, but in a "separated" configuration where OpenAKC is running on dedicated servers, it will continue recording.

[1] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/recording_sessions/index

Examples

Setting up a demo/test environment on Debian/Ubuntu using the script provided to automate configuration.

The script located in OpenAKC/contrib/debian-lxc-build+demo.sh will configure a basic environment for testing and/or demonstration purposes.

Eg:-

```
root@trinity:/home/james/GIT/OpenAKC/contrib# ./debian-lxc-build+demo.sh
Setting up OpenAKC demo environment, using standard LXC containers.

Checking for installed package, lxc - Found!
Checking for installed package, uidmap - Found!
Checking for installed package, bridge-utils - Found!
Checking for installed package, debootstrap - Found!
Checking for installed package, dnsmasq-base - Found!
Checking for installed package, gnupg - Found!
Checking for installed package, iproute2 - Found!
Checking for installed package, iptables - Found!
Checking for installed package, lxc-templates - Found!
Checking for installed package, lxcfs - Found!
Checking for installed package, openssl - Found!
Checking for installed package, rsync - Found!
Checking for installed package, acl - Found!

All Components found, continueing.

Containers will be "openakc-combined" & "openakc-client".

"openakc-combined" will contain the OpenAKC server, and user
access host (and be used to build the packages)

"openakc-client" will be the demo client system for OpenAKC
to grant access

The next step will destroy the test containers and rebuild!

Press ENTER to rebuild test containers, or ^C to abort


Destroying old containers...

Installing with options "-d ubuntu -r eoan -a amd64", please wait...

Almost done!...

NAME             STATE    AUTOSTART GROUPS IPV4 IPV6 UNPRIVILEGED
RH6-build        STOPPED 0          -      -    -    false
RH6-tmp          STOPPED 0          -      -    -    false
centos8          STOPPED 0          -      -    -    false
openakc-client   RUNNING 0          -      -    -    false
openakc-combined RUNNING 0          -      -    -    false
syslog-test      STOPPED 0          -      -    -    false

Waiting for new containers to settle

Setting up containers
```

A lot of output is generated at this point as the containers are set up, and packages compiled, so it is not included here.

After the containers are built, packages compiled and installed the user is asked to enter pre-defined pass phrases into the OpenAKC registration tool, before the example role definition from the contrib folder is applied to a target application user on the demo client machine.

```
Using openakc setrole (as admin-user) to upload the example role configuration
openakc setrole app-user@openakc-client /tmp/examplerole
NB: use "openakc editrole app-user@openakc-client" for interactive configuation

OpenAKC Copyright (C) 2019-2020  A. James Lewis.  Version is 0.99~ubuntu19.10-4.

This program comes with ABSOLUTELY NO WARRANTY; see "license" option.
This is free software, and you are welcome to redistribute it
under certain conditions; See LICENSE file for further details.

Connected to OpenAKC server.  Sending role update request
OK: Request processed, role data updated

Done!

You should now connect to the "openakc-combined" container,
Then verify that the "normal-user" account can successfully connect
using "ssh app-user@openakc-client"

You should now have a working demo/sample install in the containers!

To access the container, type "lxc-attach -n openakc-combined"
then, "su - normal-user"
then, "ssh app-user@openakc-client"

If everything above worked, you should be able to connect using the "normal-user" key (Use passphrase: "userkey")

root@trinity:/home/james/GIT/OpenAKC/contrib# lxc-attach -n openakc-combined
```

For this example, an additional user is added to demonstrate restrictions which could be applied to the root user, this user creates an ssh key pair for that user, registers with the OpenAKC server then returns to the admin-user account to create a role configuration for the demo.

```
root@openakc-combined:/# useradd -c "Restricited User" -k /etc/skel -s /bin/bash -m restricted-user
root@openakc-combined:/# su - restricted-user
restricted-user@openakc-combined:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/restricted-user/.ssh/id_rsa):
Created directory '/home/restricted-user/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/restricted-user/.ssh/id_rsa.
Your public key has been saved in /home/restricted-user/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:RCuYezJs2j+TVR40NrKQq8uDqrBVKGcI1M39p+YEa84 restricted-user@openakc-combined
The key's randomart image is:
+---[RSA 3072]----+
| .. o ...        |
|.  . =oo..=      |
|.   o .o+= o     |
|.. o ..+..o.     |
|o + B.. Soo.     |
| + =.+ o.+.      |
|. oo..+o+        |
|.o. +.+E .       |
|=.   ..o         |
+----[SHA256]-----+
restricted-user@openakc-combined:~$ openakc register
OpenAKC Copyright (C) 2019-2020  A. James Lewis.  Version is 0.99~ubuntu19.10-4.

This program comes with ABSOLUTELY NO WARRANTY; see "license" option.
This is free software, and you are welcome to redistribute it
under certain conditions; See LICENSE file for further details.

Passphrase is requested to ensure you own this key.

Enter passphrase:

Escalating to perform API call

Connected to OpenAKC server.  Sending key registration request
OK: Request processed

restricted-user@openakc-combined:~$ exit
logout
root@openakc-combined:/# su - admin-user
admin-user@openakc-combined:~$ openakc editrole app-user@openakc-client
```

Creating the role configuration was done using "editrole" which spawns an editor (Based on the EDITOR environment variable), as shown, with the role definitions applied for the demo.

```
# Edit this file to control permissions managed by OpenAKC.
# Rules for root@openakc-client.
#
# First matched rule will apply.
# {TYPE} can be user, group or key(ssh key fingerprint)
# FROM= can be either "any" or a comma separated list of IPs or CIDR subnets
# DAY= can be either "any" or a comma separated list of 3 letter day codes.
# TIM= is daily {Start Time},{End Time} 24 hour format, local server time.
# SHELL= users shell (could make shell "rjoe/rvim filename", or /bin/false).
# CMD= is a list of permitted commands, "any" or "comma separated list"
#    NB: 1) if CMD is not "any", it must include scp to allow SCP.
#        2) if CMD is not "any", sftp-server requires full path.
#        3) if CMD is exactly "internal-sftp", OpenAKC session will be
#           bypassed and "internal-sftp" will be forced.  Care! No capability
#           restrictions can be applied in this mode, and no session
#           recording will take place other than sftp's own logging.
#        4) Internal SFTP can be chrooted in sshd_config, while capabilities
#           can be applied to external sftp Eg (/usr/lib/openssh/sftp-server).
#        5) if a users calls an absolute path to a binary then CMD must
#           match the absolute path, otherwise short names are OK.
#    EG: CMD=scp,ls,uname,dmidecode,/usr/lib/openssh/sftp-server
# SCP= "sed" expression implementing chroot for scp connections.
# CAP= list of linux capabilities denied to role (man capabilities).
# REC= record session log, yes/no
#
# Eg: (Warning, TAGs must be CAPITAL, do not use quotes)
#
#ROLE=2020/01/13 19:17,2020/01/13 20:17,{TYPE},jlewis
#DAY=any
#TIM=any
#SHELL=/bin/bash
#CMD=any
#SCP=s,^/,/data/,g
#CAP=cap_linux_immutable
#REC=yes
#FROM=any
#
ROLE=2020/01/13 19:17,2025/01/13 20:17,user,normal-user
DAY=any
TIM=any
SHELL=/bin/bash
CMD=any
SCP=s,^/,/tmp/,g
CAP=cap_linux_immutable
REC=yes
FROM=any

ROLE=2020/01/13 19:17,2025/01/13 20:17,user,restricted-user
DAY=any
TIM=any
SHELL=/bin/bash
CMD=any
SCP=s,^/,/tmp/,g
CAP=cap_linux_immutable,cap_net_raw+ep
REC=yes
~
~
~
~
```

The "CAP=" field lists capabilities removed from that role, and you will note that the "restricted user" has an extra capability denied.

Now we log in to each of "normal-user" and then "restricted-user", useing the roles thus configured to access the "root" account on the target, and show that this role definition blocks access to raw network, even for the "root" account.

```
root@openakc-combined:/# su - normal-user
normal-user@openakc-combined:~$ ssh root@openakc-client
Enter passphrase for key '/home/normal-user/.ssh/id_rsa':
OpenAKC Interactive Session Initialized

Please enter the reason for your connection
Title / Change Number: 1234
Description (blank line to end)
Test restrictions

Restrictions Applied:-
Captbility cap_linux_immutable withdrawn by role config

root@openakc-client:~# tcpdump -ni eth0 port 25
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
root@openakc-client:~# exit
exit
Connection to openakc-client closed.
normal-user@openakc-combined:~$ exit
logout
root@openakc-combined:/#
root@openakc-combined:/#
root@openakc-combined:/#
root@openakc-combined:/# su - restricted-user
restricted-user@openakc-combined:~$ ssh root@openakc-client
Enter passphrase for key '/home/restricted-user/.ssh/id_rsa':
OpenAKC Interactive Session Initialized

Please enter the reason for your connection
Title / Change Number: 1234
Description (blank line to end)
Test restrictions

Restrictions Applied:-
Captbility cap_linux_immutable withdrawn by role config
Captbility cap_net_raw+ep withdrawn by role config

root@openakc-client:~# tcpdump -ni eth0 port 25
tcpdump: eth0: You don't have permission to capture on that device
(socket: Operation not permitted)
root@openakc-client:~# exit
exit
Connection to openakc-client closed.
restricted-user@openakc-combined:~$ exit
logout
root@openakc-combined:/#
```

Session logs for the restricted user record the event:-

```
root@openakc-combined:/# cd /var/lib/openakc/keylogs/2020/07/03/
root@openakc-combined:/var/lib/openakc/keylogs/2020/07/03# ls -lastr
total 60
 4 drwxr-xr-x 3 root root  4096 Jul  3 07:45 ..
 8 -rw-r--r-- 1 root root  4929 Jul  3 07:45 07:45:13-root@openakc-client.restricted-user.1348-000a0803.log
 4 -rw-r--r-- 1 root root   128 Jul  3 07:46 07:46:07-root@openakc-client.restricted-user.1794-000a0803.log
 4 -rw-r--r-- 1 root root   871 Jul  3 07:51 07:51:19-root@openakc-client.normal-user.2170-000a0803.log
 4 -rw-r--r-- 1 root root   128 Jul  3 07:53 07:52:59-root@openakc-client.normal-user.2543-000a0803.log
 4 -rw-r--r-- 1 root root   128 Jul  3 07:57 07:57:36-root@openakc-client.normal-user.2916-000a0803.log
 4 -rw-r--r-- 1 root root   128 Jul  3 08:00 08:00:13-root@openakc-client.restricted-user.3288-000a0803.log
20 -rw-r--r-- 1 root root 17618 Jul  3 08:32 08:32:16-root@openakc-client.restricted-user.3676-000a0803.log
 4 -rw-r--r-- 1 root root   710 Jul  3 08:37 08:34:51-root@openakc-client.restricted-user.4051-000a0803.log
 4 drwxr-xr-x 2 root root  4096 Jul  3 08:37 .
root@openakc-combined:/var/lib/openakc/keylogs/2020/07/03# cat 08:34:51-root@openakc-client.restricted-user.4051-000a0803.log
Title: 1234
Summary: Testing Restrictions
Summary:

Network Source: 10.0.3.217 55518 22
Key Used: SHA256:Tar9D8z4ok7BwYwTq5t4mtLTnEcuoecK0Dn04kqVcd0
TTY: /dev/pts/4
Term Type: xterm-256color
Term Language: en_US.UTF-8
Shell: /bin/bash
Restricted Capabilties: cap_linux_immutable,cap_net_raw

Script started on 2020-07-03 08:34:51+00:00 [TERM="xterm-256color" TTY="/dev/pts/4" COLUMNS="254" LINES="68"]
root@openakc-client:~# tcpdump -ni eth0 port 25
tcpdump: eth0: You don't have permission to capture on that device
(socket: Operation not permitted)
root@openakc-client:~# exit
exit

Script done on 2020-07-03 08:35:13+00:00 [COMMAND_EXIT_CODE="1"]
root@openakc-combined:/var/lib/openakc/keylogs/2020/07/03#
```

Another example of useful restrictions which could be applied using OpenAKC's capability function would be protecting sensitive user files from the root user. You may want to have a 1st line support group who do not have full access to the system, and in this example, you can see the power of the tool, as it blocks "root" from accessing a users files, by blocking root's ability to override file permissions… as well as it's ability to change UID/GID via tools such as "su".

```
normal-user@openakc-combined:~$ ssh root@openakc-client
Enter passphrase for key '/home/normal-user/.ssh/id_rsa':
OpenAKC Interactive Session Initialized

Please enter the reason for your connection
Title / Change Number: 1234

Description (blank line to end)
Test Restrictions

Restrictions Applied:-
Captbility cap_linux_immutable withdrawn by role config
Captbility cap_setgid withdrawn by role config
Captbility cap_setuid withdrawn by role config
Captbility cap_chown withdrawn by role config
Captbility cap_dac_override withdrawn by role config
Captbility cap_dac_read_search withdrawn by role config
Captbility cap_fowner withdrawn by role config

root@openakc-client:~# cd /home/app-user/.ssh/
bash: cd: /home/app-user/.ssh/: Permission denied
root@openakc-client:~#
root@openakc-client:~# su - app-user
su: cannot set groups: Operation not permitted
root@openakc-client:~#
root@openakc-client:~# chown root:root /home/app-user/.ssh
chown: changing ownership of '/home/app-user/.ssh': Operation not permitted
root@openakc-client:~#
root@openakc-client:~# ls -asl /home/app-user/.bashrc
4 -rw-r--r-- 1 app-user app-user 3771 May  5  2019 /home/app-user/.bashrc
root@openakc-client:~# echo "Modified" >> /home/app-user/.bashrc
bash: /home/app-user/.bashrc: Permission denied
root@openakc-client:~#
root@openakc-client:~#
root@openakc-client:~#
root@openakc-client:~# echo "Modified" >> /etc/hosts
root@openakc-client:~#
root@openakc-client:~# reboot
Connection to openakc-client closed by remote host.
Connection to openakc-client closed.
normal-user@openakc-combined:~$
```

Of course, you may want to further protect the user data, by blocking access to raw disk IO, but remember, that these restrictions will eventually impact normal operation. For example, a "root" user blocked from accessing user files can create entries in /etc/passwd, /etc/shadow etc…. So could create a new user, but could not then create that users home folder. Always test new restrictions to understand where they may not be suitable.

In some other examples, session logs are able to show both non-interactive commands, and scp file transfers, as in this example, where we can see that the user executed a remote (non-interactive) command "uname -a", and how this is recorded in the session log.

```
noping-user@openakc-combined:~$ ssh app-user@openakc-client 'uname -a'
Enter passphrase for key '/home/noping-user/.ssh/id_rsa':
Linux openakc-client 5.4.0-8-generic #11-Ubuntu SMP Fri Dec 6 22:43:53 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
noping-user@openakc-combined:~$ exit
logout
root@openakc-combined:/# cat /var/lib/openakc/keylogs/2020/07/03/03\:29\:07-app-user\@openakc-client.noping-user.2533-000a2603.log
"uname -a" called via ssh

Script started on 2020-07-03 03:29:07+00:00 [<not executed on terminal>]
Linux openakc-client 5.4.0-8-generic #11-Ubuntu SMP Fri Dec 6 22:43:53 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux

root@openakc-combined:/#
```

As well as SCP transfers. Note how the target is changed from /myfile to /tmp/myfile by the search/replace "sed" script in the role configuration.

```
normal-user@openakc-combined:~$ scp /etc/hosts app-user@openakc-client:/myfile
Enter passphrase for key '/home/normal-user/.ssh/id_rsa':
hosts                                                          100%  182   664.7KB/s   00:00
normal-user@openakc-combined:~$ exit
logout
root@openakc-combined:/# cat /var/lib/openakc/keylogs/2020/07/03/03\:
03:29:07-app-user@openakc-client.noping-user.2533-000a2603.log  03:38:12-root@openakc-client.normal-user.3257-000a2603.log
03:37:47-root@openakc-client.normal-user.2884-000a2603.log      03:39:22-app-user@openakc-client.normal-user.3629-000a2603.log
root@openakc-combined:/# cat /var/lib/openakc/keylogs/2020/07/03/03\:39\:22-app-user\@openakc-client.normal-user.3629-000a2603.log
"scp -t /myfile" called via ssh

SCP PUT /tmp/myfile executed, exit code 0
```

And into the local log on the client machine.

```
root@openakc-client:~# grep scp /var/log/syslog
Jul  3 03:39:22 openakc-client openakc-scp: PUT /tmp/myfile ; Succeeded for user app-user
root@openakc-client:~# exit
exit
Connection to openakc-client closed.
normal-user@openakc-combined:~$ exit
```

Appendices

1. Command Reference & Examples

Command "openakc-plugin"

The tool "openakc-plugin" is primarily the interface between the ssh daemon,on the client machine and the OpenAKC backend server via the API. It does however have a couple of additional functions which are are useful for debugging issues.

1. When used as a plugin, it is called in the "sshd_config" as follows (note that the installation process will update the ssh configuration so you should not need to do this manually):-

AuthorizedKeysCommand /usr/sbin/openakc-plugin %u %h %f

```
AuthorizedKeysCommandUser openakc
```

The parameters passed to the plugin include the local user and home folder of the local user for which the incoming connection wishes to be authenticated, as well as the key fingerprint of the key being used as a token. Also, notice that the client runs as an unprivileged user.

2. When running as a normal user (not root), the only other possible use of this tool is to test connectivity to the server:-

```
$ openakc-plugin ping
```

The server response "OK: Pong!" will be seen if connection is working OK.

3. When running as root there is one additional option available, this is to reset the server public keys stored on the client. This might be required if the servers identity changes. In this scenario, the client will no-longer communicate with the server until the keys are reset. This is similar to the way SSH itself will warn before connecting to a server who's host keys have changed:-

```
$ openakc-plugin resetkeys
```

Command "openakc"

The tool "openakc" is the tool used by both users, and admins to interact with the API from the command line.

Users

OpenAKC is largely intended to be transparent to the user, at the point of use, so with the exception of restrictions imposed by the role rules (or the login reason "quiz"), the only real reason for a user to interact with AKC is to register their public key.

```
$ openakc register
```

This verifies the ownership of the public key and identity of the user, then records this information with the backend server via the API. Once recorded in this way the users public key will be made available on client machines, wherever they wish to log in, so long as the role rules governing access to that system permit them to log in.

Eg. If a new web developer "fred" joins, assuming his user account in AD (or LDAP etc.) is created as a member of the group "webadmins", then provided there is a rule permitting members of the "webadmins" group to login in via ssh to "apache@webserver", then "fred" only needs to create and register a suitable ssh key pair (pass phrases are mandatory), and he will instantly be able to log in to that account. There is no delay while keys are distributed, since keys are not distributed but are accessed on demand by the client as part of the authentication process.

Admins

Security administrators have two functions they can perform via the "openakc" tool.

1. Defining role rules for access to given clients, this can be done either via an editor (defined by the EDITOR environment variable).

```
$ openakc editrole username@hostname
```

Alternatively, they can download the existing roles, or upload a pre-defined set of rules.

```
$ openakc getrole username@hostname filename
```
or
```
$ openakc setrole username@hostname filename
```

The "getrole" command will overwrite file defined by "filename" with the current rules for the specified account, while the "setrole" command will upload the content of "filename" to replace the existing rules.

2. Managing static trust (equivalent to authorized_keys).

Public keys can be uploaded to the system using the "submit" command (pass phases are not enforced for this mode, but obviously recommended where possible). The openakc tool will provide a serial number when a key is submitted.

```
$ openakc submit filename "Comment Text"
```

```
admin-user@openakc-combined:~$ openakc submit /tmp/id_demo "Demo Key"
OpenAKC Copyright (C) 2019-2020  A. James Lewis.  Version is 0.99~ubuntu20.04-7.

This program comes with ABSOLUTELY NO WARRANTY; see "license" option.
This is free software, and you are welcome to redistribute it
under certain conditions; See LICENSE file for further details.

Storing static key /tmp/id_demo with comment "Demo Key"
Connected to OpenAKC server.  Sending key submission request
OK: Stored static key - S/N = key-kIbjBGBvnO
```

Existing public keys can be updated without changing the serial number using the "update" command. The openakc tool will overwrite the key with the specified serial number, which is useful, because rules can refer to keys by serial number, thus allowing keys to be replaced/rotated without updating all the rules which might refer to them.

```
$ openakc update key-serial filename "Comment Text"
```

As a subset of the update feature, a key can be removed from the server as follows.

```
$ openakc update {key-serial|username} DELETE
```

Note that the delete feature can also be used for user keys, perhaps when a member of staff leaves and there account is removed. Obviously deleting a users key would be a housekeeping function, as their key would not be useful after their account was deleted in whatever directory was used, but for static trust relationships it would definitely be easier to delete a key than search all the rules to see if it is referenced.

2. API Reference & Examples.

The API has 2 stages. Stage one is unencrypted, and allows the client to establish the connection, and negotiate the encryption, while stage 2 is where the business of of exchanging credentials and performing authentication happens.

Stage 1 Options.

help

ping

getproto

setproto

quit

Stage 2 Options.

Note that for debugging purposes, stage 2 commands can be sent unencrypted, or sent with encryption using the "message" command.

help

pubkey

sessioncode

challenge

handshake

setrole

getrole

listrole

registerkey

submit

update

sessiondata

message

audit

auth

ping

logturn

quit